Introduction

Security is a journey not a destination. What’s been deployed today may be found to have a vulnerability tomorrow. What you thought was a trusted source may be a bad actor trying to compromise your IT systems. How can you ensure that your organization is protected from the full gamut of application security pitfalls?

Today, many organizations are turning to Docker Enterprise Edition (EE) and Windows Server 2016 to build a platform that allows for the delivery of safer applications. Applications that they know come from secured sources and are up to date with the latest security patches.

When looking at application containers and the security surrounding them, Docker believes there are three key characteristics that directly correlate to safer apps:

**Usable Security**: Security features need to be usable by people at all stages of a software supply chain. It’s been proven time and time again, people will actively circumvent overly onerous security policies and procedures. Both Docker Enterprise Edition and Windows Server 2016 are designed to be secure by default, but are also built with usable tooling that makes sense for developers and operators – enabling workflows that work for them.

**Trusted Delivery**: By their very nature applications move around. Because of this, individuals need to ensure that their code can move safely from point A to point B with proof that nobody has tampered with it. Docker Enterprise Edition and Windows Server 2016 provide key features to ensure the integrity of the applications that you deploy into production.

**Infrastructure Independent**: The promise of Docker is application portability. But, this portability is not limited to the application code itself. With Docker, security configurations are intrinsic to the application, and can move from a developer’s workstation through testing to production deployment - whether that’s Windows Server 2016 running in Azure or your datacenter - without any modification all managed via Docker EE.

### The Key Components of Container Security

<table>
<thead>
<tr>
<th>Usable Security</th>
<th>Secure defaults with tooling that is native to both dev and ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted Delivery</td>
<td>Everything needed for a full functioning app is delivered safely and guaranteed to not be tampered with</td>
</tr>
<tr>
<td>Infrastructure Independent</td>
<td>All of these things in your system are in the app platform and can move across infrastructure without disrupting the app</td>
</tr>
<tr>
<td>Safer Apps</td>
<td></td>
</tr>
</tbody>
</table>
Usable Security

Usable security is defined by two key components: secure defaults and tooling that works for both development and operations teams. Overly complex security policies and technologies are inevitably rendered ineffective as people find ways to work around them rather than with them. With Windows Server 2016 and Docker EE, security features are designed to provide optimal application security (including protecting sensitive data) while integrating easily into existing workflows for both developers and operations professionals.

Secure by Default Runtime

Docker containers have become a critical component for developers in organizations of any size. Windows Server 2016 ships with support for Windows Server containers. Windows Server containers are powered by Docker Enterprise EE, and are the result of over 2 years of joint engineering effort between Microsoft and Docker.

Windows Containers

Windows Server containers provide isolation for key system resources including the Windows namespace, system processes, filesystem and registry.

Registry and Filesystem

Each container writes changes to its own instance of the registry and filesystem, isolating it from the host and other containers.

Namespace

Instead of just allowing changes to the state within the container, namespace isolation on Windows renders most privileged APIs unusable. Drivers cannot be loaded in a container, and therefore cannot change the available attack surface. Containers have their own separate system services such as LSASS, service control manager, and task scheduler to provide common Windows functionality inside the container.
**Users**

Separation between well-known Windows privilege levels including `LocalSystem`, `Administrators`, and `Users` are preserved in containers. By default, all processes are also created as `ContainerAdministrator` instead of the `LocalSystem` process to separate privilege by default.

**Resources**

Resource limitations can be applied for CPU, memory, disk usage, and disk throughput for each Windows Server container too. These tools let you protect the performance of your important applications running in containers while still driving up overall utilization.

**Hyper-V Isolation**

Windows Containers with Hyper-V Isolation

Hyper-V isolation adds another layer of isolation by putting the container in a specially optimized Hyper-V virtual machine with a completely separate Windows kernel instance. You can start a container with Hyper-V isolation using a normal `docker run` command and adding the `--isolation=hyperv` option.

For example:

```
docker run --isolation=hyperv -d -p 8000:8000 --name mysite iis-site
```

will start a copy of an IIS-based web site using Hyper-V isolation.

You do not need to take any extra steps to setup a VM, configure it, or remove it later. Everything is automatic and optimized to run your container. You can even run containers with and without Hyper-V isolation on the same machine to deploy each individual container with the level of security it needs. Additionally, there is no change needed to Docker images for them to work with Hyper-V isolation. Docker images work the same with both Hyper-V isolation as well as standard Windows Server containers. Hyper-V isolation make it easy to meet your organization's security and regulatory requirements without adding complexity.
Application Secrets

When deploying and orchestrating services, administrators often need to configure those workloads with sensitive information like passwords, TLS certificates, or private keys.

Docker Enterprise Edition provides IT professionals the ability to store this sensitive information, also known as secrets, in a secure way.

In terms of Docker Enterprise Edition, a secret is a blob of data, such as a password, SSH private key, SSL certificate, or another piece of data that should not be transmitted over a network or stored unencrypted in a Dockerfile or in your application’s source code. With Docker Enterprise Edition, you can use Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it. Secrets are encrypted during transit and at rest. A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

Trusted Security

Ultimately IT professionals want to know that the software they have deployed in their environments comes from trusted sources, and that their code is free from vulnerabilities. But, as often is the case, this is easier said than done. However, with Docker Enterprise Edition and Windows Server 2016 administrators can set policies to ensure only trusted code is deployed. And that code can be scanned and verified throughout the lifecycle of the application.

Docker Image Signing and Verification

Knowing that what’s running in your environment at any given time came from a trusted source is critical to protecting your information assets. But, how can you ensure the images being deployed in your organization are coming from a trusted source? One way is to have those images digitally signed. In the past content signing has been a complex problem, and existing solutions were not well suited to containers.

Docker solves the challenges associated with image signing and verification of containers via Docker Content Trust (DCT). Unlike PGP and other tools, DCT has been purpose-built to make the software supply chain secure and auditable. Docker Content Trust uses The Update Framework (TUF), a language-agnostic software update system, and is implemented in Notary, an open source tool included with Docker Enterprise Edition which provides trust over any content.

The core principle in Docker Content Trust is that each principal should sign the image when they are done handling it. For example, when Security Scanning is complete, the Scanner can sign a statement asserting that fact. Before each step in the pipeline, the responsible party will also verify all signatures. A container that has been tampered with will be detected because either its hash will not match the signatures, or it will be missing signatures from each of the necessary parties.

Signatures for Docker Content Trust are stored in a separate metadata store to promote separation of concerns. Because the system derives its security from the cryptography that it uses, the metadata store itself does not have to be trusted. Instead, each system and repository has its own keys. These use the trust-on-first-use principle by default, but additional verification is possible.

Docker Security Scanning

As previously mentioned, security doesn’t stop when an application is deployed. It’s important to keep track of any new vulnerabilities that are discovered as they may affect currently running applications. This can be a time-consuming task without some level of automation.

Thankfully Docker Trusted Registry (DTR), a core component of Docker Enterprise Edition, automates this process.
Docker Trusted Registry can scan images in your repositories to verify that they are free from known security vulnerabilities or exposures, using Docker Security Scanning. The results of these scans are reported for each image tag.

The Docker Security Scan process is straight-forward, and thorough. First, the scanner performs a binary scan on each layer of the image, identifies the software components in each layer, and indexes the SHA of each component. A binary scan evaluates the components on a bit-by-bit level, so vulnerable components are discovered no matter what they’re named, even if they’re statically-linked, and regardless of if they’re included on a distribution manifest.

The scan then compares the SHA of each component against the Common Vulnerabilities and Exposures (CVE®) database installed on your DTR instance. The CVE database is a “dictionary” of known information security vulnerabilities. When the CVE database is updated, the service reviews the indexed components for any that match newly discovered vulnerabilities.

Docker Security Scanning can be configured to run automatically on `docker push` to an image repository.

If your DTR instance is configured in this way, you do not need to do anything once your `docker push` completes. The scan runs automatically, and the results are reported in the repository’s Images tab after the scan finishes.

Finally, only users with write access to a repository can manually start a scan. Users with read-only access can view the scan results but cannot start a new scan.

**Windows Updates**

Working alongside Docker Security Scanning, Microsoft Windows Updates can ensure that your Windows Server operating system is always up to date. Microsoft publishes two pre-built Windows Server “base” operating system containers - `microsoft/nanoserver` and `microsoft/windowsservercore` on the Docker Store. These images will be updated the same day as new Windows security updates are released. Rather than going through the hassle of maintaining their own base images, IT practitioners can build their projects on top of the official Microsoft images and rest assured they have the latest security updates. All of this makes it easy to bring the official Windows Server images into existing continuous delivery and deployment workflows and include Windows updates in the same process as all your other deployments.

**Windows Code Integrity**

Windows Code Integrity policies cryptographically validate all Windows system processes in containers with the same policy applied to the host. This ensures the Windows container images are genuine and prevents tampered Windows binaries from running in containers.

Anti-malware products, including Windows Defender on Windows Server 2016 and Windows 10, scan all containers as they are pulled and extracted onto the host. Containers containing malware can be detected and blocked before they are ever executed.

**Infrastructure Independence**

The promise of Docker is that your application can move seamlessly between environments: from dev to QA to production or from a VM to bare metal to a cloud instance. With Docker Enterprise Edition and Windows Server 2016 you can also rest assured that not only does your application move seamlessly, but so do its security definitions. Additionally, Docker Enterprise Edition integrates into your existing Windows Server environment.

**Using Active Directory Service Accounts for Containerized Application Identity**

Active Directory is the built-in service used for discovery, search and replication of user, computer, and service account
information on Windows. It is commonly used for authentication and authorization between users and services, or between services on different machines. This is a simpler and more secure way and simpler to manage than locally stored usernames & passwords because the Active Directory provides a single place to handle authorization, revocation, and password rollovers.

Servers are typically Active Directory domain-joined today, which gives each server a unique identity in the Active Directory domain and enables domain-joined services. Services running as Local System or Network Service can authenticate connections using the server’s domain account. Windows containers can use a similar mechanism to run as an Active Directory Group Managed Service Account. This enables applications running in a container such as a website to authenticate with other services such as a backend SQL Server using an Active Directory identity. There is no password or certificate private key stored in the container image that could be inadvertently exposed, and the container can be redeployed to development, test, and production environments without being rebuilt to change stored passwords or certificates.

Single Sign-On with Active Directory

Not only can Active Directory be used for managing application credentials inside containers, it can also be used to provide user and group synchronization with Docker Enterprise Edition. Docker Enterprise Edition allows for single sign-on between Docker Trusted Registry and Docker Universal Control Plane. User accounts can be managed locally, or they can be synchronized with your Active Directory.

Role-based Access Control

Docker Enterprise Edition allows administrators to apply fine-grained role based access control to a variety of Docker primitives, including volumes, nodes, networks, and containers. IT operators can grant users predefined permission roles to collections of Docker resources. Docker EE also provides the ability to create custom permissions roles, providing IT operators tremendous flexibility in how they define access control policies in their environment.

Conclusion

As was stated at the beginning of this paper, security is a journey not a destination. A system that is secure today, may not be secure tomorrow. That’s why IT professionals need systems that provide not only the highest level of security, but are also designed to ensure the integrity of the applications that run on them from the moment they are deployed until the moment they are retired.

Windows Server 2016 and Docker Enterprise Edition together offer an unparalleled suite of features to ensure that information resources are protected continuously from the moment they are deployed.

Additional Resources

Learn more about Docker Enterprise Edition at https://www.docker.com/enterprise or test drive it for yourself at https://www.docker.com/trial. Or you can contact sales at https://www.docker.com/contact.